Learning to Sample Graphs from Partially Observed Cliques

Edward Hu [*1] **Cristian Dragos Manta** [*1] **Hadi Nekoei** [*1] **Yoshua Bengio** [1]

1. Introduction

Probabilistic models with latent variables are a powerful tool when the structure of the latents takes on a form that makes the conditional distribution $\mathbb{P}(x \mid h)$ easier to model than the joint distribution $\mathbb{P}(x, h)$. A fundamental problem in reasoning is to devise such a latent structure h that adequately explains the underlying dynamics of the high-dimensional sensory input x. Findings in cognitive science and neuroscience hint at a low-dimensional and sparsely connected latent structure, which has inspired myriad inductive biases for deep learning models (Goyal & Bengio, 2022). For example, we can parametrize the energy function of latent and observable variables as a Markov network (Sherrington & Kirkpatrick, 1975), with latent variables being abstract concepts and cliques of concepts being thoughts or mechanisms. We will call this the **simplified thinking machine** and work with such a parametrization throughout this work. More generally, the desire to directly learn such latent structures in a probabilistic framework and to sample them motivated generative flow networks (GFlowNets) (Bengio et al., 2021a).

GFlowNets were originally framed as a modified soft-Q learning algorithm (Haarnoja et al., 2018) from reinforcement learning (RL) that properly deal with multiple trajectories ending in the same final state with applications in drug discovery. The GFlowNet framework was then formalized as a general-purpose amortized sampler of distributions over structured objects (Bengio et al., 2021b). It is unique in both the sequential nature of its sampling process and the internal consistency objectives, such as detailed balance (Bengio et al., 2021b) and trajectory balance (Malkin et al., 2022), it optimizes, both of which are inspired by RL.

One crucial limitation to the current GFlowNet framework is the need to have complete trajectories. The internal consistency objectives used to train GFlowNets distinguish between flows between two internal states and flows between an internal state and a terminal state. Only the latter are a function of the external reward or energy function, meaning that if we never visit those terminal transitions, we will never obtain learning signals. While typically not an issue for applications such as drug discovery, this becomes a crippling limitation when learning to sample graphs with millions of latent variables, as in the case of the simplified thinking machine. The primary problem is that we might never visit a terminal transition, i.e., we never sample all the values required to evaluate the Markov network energy function and always marginalize over those unsampled values. Intuitively, this means we only "think" about concepts that are relevant to a particular input while marginalizing over irrelevant concepts.

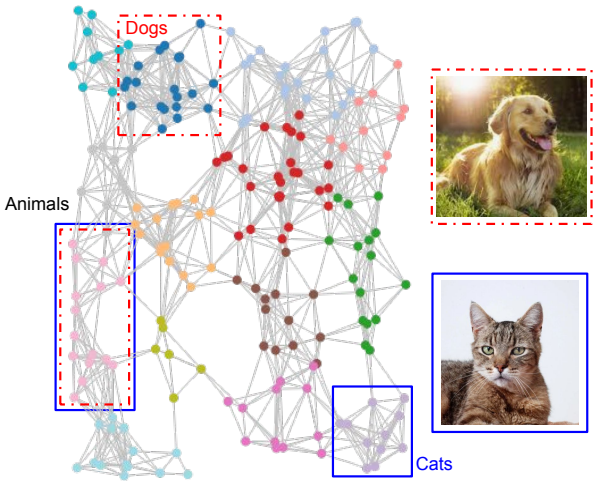


Figure 1. An illustration of the simplified thinking machine on image data. The complete Markov network is enormous, yet an individual input x only shares high mutual information with a sparse set of cliques $z_{z_x} \subset C(z)$, where $C(z)$ is the set of maximal cliques in z. We use synthetic data to study a new training objective that does not require observing all cliques.

In this work, we focus on learning to sample values for a given concept while deferring the problem of selecting what concepts are relevant to future work. A consequence of our simplification is that our results are still currently obtained over complete trajectories. However, our model does not receive any reward that is associated to the fact that a transition is terminating, as we will see in section 2. The key technical question is: can we exploit modularity

[*]Students of IFT6269. Ordered by when one joined the project. [1]DIRO, Université de Montréal. Correspondence to: Edward Hu <edward.hu@mila.quebec>, Cristian Dragos Manta <cristian-dragos.manta@mila.quebec>, Hadi Nekoei <nekoeihe@mila.quebec>.

Course Project Report for IFT6269, Dec 2022.

in the energy function, e.g., a Markov network with many maximal cliques, and derive partial reward signals to train a GFlowNet to sample from the correct posterior over missing values? We answer in the affirmative and make the following contributions:

- A modified detailed balance objective that uses additive energy terms as partial reward for GFlowNet training;
- Empirical experiments demonstrating its effectiveness in a simplified setup where trajectories are still complete, but the model only receives partial rewards, as explained below.

2. Modified Detailed Balance for Modular Energy Functions

Given an energy function $\mathcal{E}(x, z)$ where x denotes observable variables and z latent variables, a conditional GFlowNet that takes x as input learns to sample from

$$\mathbb{P}(z \mid x) = \frac{\mathbb{P}(z,x)}{\mathbb{P}(x)} \propto \mathbb{P}(z,x) = \frac{\exp[-\mathcal{E}(x,z)]}{Z} \propto \exp[-\mathcal{E}(x,z)]$$

where $Z = \sum_{x,z} \exp[-\mathcal{E}(x,z)]$.

Following the parametrization in Bengio et al. (2021b), we use P_F to denote the forward policy of a GFlowNet, P_B the backward policy, and F the state flow function. The detailed balance condition states that if we have

$$F(s \mid x)P_F(s' \mid s,x) = F(s' \mid x)P_B(s \mid s',x) \quad (1)$$
$$F(s_z \mid x)P_F(\top \mid s_z,x) = R(s_z,x) = \exp[-\mathcal{E}(s_z,x)] \quad (2)$$

for all adjacent state pairs (s, s') and all terminating state s_z where $R(s_z, x)$ is defined and non-negative, we will obtain a sampler over terminating states $\sum_{\tau \in T(s_z)} P_F(\tau) = \frac{\exp[-\mathcal{E}(s_z,x)]}{\sum_{s'_z} \exp[-\mathcal{E}(s'_z,x)]}$. We have used $T(s_z)$ to denote all trajectories ending in state s_z and $\top$ the terminating action. Concretely, s_z will contain a full instantiation of all the latent variables z, which is required to calculate the corresponding energy $\mathcal{E}(z, x)$, where a non-terminating state s could have a partial instantiation of variables in z.

To satisfy the detailed balance condition, we minimize the following mean-squared-error losses in the log domain:

$$\mathcal{L}(s, s' \mid x) = (\log F(s \mid x) + \log P_F(s' \mid s,x) \quad (3)$$
$$- \log F(s' \mid x) - \log P_B(s \mid s',x))^2 \quad (4)$$
$$\mathcal{L}(s_z \mid x) = (\log F(s_z \mid x) + \log P_F(\top \mid s_z,x) \quad (5)$$
$$+\mathcal{E}(s_z,x))^2 \quad (6)$$

However, if we never visit a terminating state s_z, where all variables have been instantiated with a value, we never call the energy function $\mathcal{E}$ and have no chance of learning to sample according to it.

Let us consider a case where the energy function $\mathcal{E}(s_z, x)$ can be decomposed into additive modules

$$\mathcal{E}(s_z,x) = \sum_{c \in C(s_z)} \mathcal{E}(c,x) \quad (7)$$

where $C(s_z)$ denotes the set of additive modules in s_z, e.g., a set of maximal cliques each with its own energy or potential function. Remarkably, this modularity can be used to introduce learning signals that are obtained after the full instantiation of each module instead of after the full instantiation of z.

Definition 2.1. Given a flow function $F(s)$, we define the **forward-looking** flow function

$$\widetilde{F}(s) \triangleq \exp[\mathcal{E}(s)]F(s) = \exp[\sum_{c \in C(s)} \mathcal{E}(c)]F(s) \quad (8)$$

We can plug Eq. 8 into Eq. 1 and obtain

$$\widetilde{F}(s \mid x)P_F(s' \mid s,x) = \quad (9)$$
$$\widetilde{F}(s' \mid x)\exp[-\mathcal{E}(s'-s,x)]P_B(s \mid s',x) \quad (10)$$
$$\widetilde{F}(s_z \mid x)P_F(\top \mid s_z,x) = 1 \quad (11)$$

after canceling $\exp[-\mathcal{E}(s,x)]$ on both sides. We have used $(s' - s)$ to denote the fully instantiated modules newly introduced by transitioning from s to s'. We can similarly write down the corresponding mean-squared-error losses

$$\widetilde{\mathcal{L}}(s, s' \mid x) = (\log \widetilde{F}(s \mid x) + \log P_F(s' \mid s,x) \quad (12)$$
$$- \log \widetilde{F}(s' \mid x) - \log P_B(s \mid s',x) \quad (13)$$
$$+\mathcal{E}(s'-s,x))^2 \quad (14)$$
$$\widetilde{\mathcal{L}}(s_z \mid x) = (\log \widetilde{F}(s_z \mid x) + \log P_F(\top \mid s_z,x))^2 \quad (15)$$

We note that the learning signals are provided whenever a transition newly introduces a full instantiated module, e.g., a maximal clique in a Markov network. We do not rely on the terminating state for learning anymore; in fact, the terminating transition does not even include a call to the energy function $\mathcal{E}$. We call Eq. 12 the **modified detailed balance loss for modular energy functions** and empirically validate it in the next section.

3. Empirical Validation

We use synthetic Markov networks to validate the proposed training objective. Sec. 3.1 describes the ground-truth Markov networks we use. Sec. 3.4 illustrates the metrics we use to evaluate the learned GFlowNet sampler. Sec. 3.5 presents our empirical results.

3.1. Synthetic Data Generation

For the experiments with a random graph, we generate a random undirected graph over the latent variables $z \in \mathbb{R}^d$ (with a predetermined ground truth latent space dimension) where there is an edge between each z_i and z_j according to a Bernoulli distribution with fixed parameter $p = 0.6$. We then connect all of the z_i's to an x_j for every j (the dimension of x is also part of the known ground truth parameters). Let this graph be G. We then define random energy functions $\mathcal{E}_c(x_c, z_c)$ for every clique c in G and we sample a batch of $x \in \mathbb{R}^{N \times D}$ from the resulting Markov network, where N is the batch size and D is the dimension of the observed space. Both the observed and the latent variables are binary in our setup.

Later on, we also conduct an experiment with a handcrafted graph and energy functions. In that case, the data generation method for x is the same, except that the Markov network is not randomly generated.

3.2. Model Parametrization and Training Procedure

Although in principle we would need two GFlowNets to model the mechanisms (represented by cliques) selection and the sampling of their corresponding values separately, in the context of our simplified setup, we fix a deterministic order for the selection of cliques and we learn the parameters of a single GFlowNet (that we call value policy) that is solely responsible for sampling the values of the nodes in the clique chosen by the deterministic clique policy. Moreover, the backwards transition probabilities $P_B(s|s', x)$ are fixed to 1 and our learning targets are the quantities $\log \tilde{F}$ and $\log P_F$ given in Eq. 12.

In order to learn these quantities well, we use the Transformer architecture (Vaswani et al., 2017) implemented by DeepMind in the Haiku library (Hennigan et al., 2020) as part of their example models. We decided to use this architecture because of the sequential nature of our prediction problem and since transformer-based architectures have proven to be very successful at this kind of task, namely in language modeling (Vaswani et al., 2017). We also experimented with using a GNN (Battaglia et al., 2018) (we hoped that the connectivity pattern of the sampled graphs gives additional useful features to learn from), but without achieving better results unfortunately.

The model receives as input a vector of the values of all the latent nodes whose values were previously sampled, as well as a special feature to indicate the next target latent node to sample. In addition, we condition the GFlowNet on the observation $x \in \mathbb{R}^{N \times D}$ by also passing the values of x into this vector. The future latent nodes (whose values are not sampled yet and that are not the target) receive a "dummy" feature. These features are translated into 256-dimensional embeddings to which we add positional embeddings (to identify the index of each node). The resulting new embeddings are sent to the transformer module and the output of the transformer is a new set of 256-dimensional feature vectors for each node. We select the vector corresponding to the target node and pass it to a single layer MLP that outputs the logits for each one of the possible choices of values for that node (in our case, 2). In addition, we also compute the forward-looking flow by concatenating the features of all the nodes into a single vector that we pass to a separate single layer MLP. Finally, we sample the value of the target node from a categorical distribution according to the output logits.

We repeat the sampling procedure above for each node in a clique before receiving the partial credit. This entire loop over the clique is what makes a single transition from the point of view of the GFlowNet.

For the training procedure, each trajectory starts with the empty state and ends when all the latent variables values are sampled. We run the GFlowNet for the first 500 iterations without receiving any training signal to fill the replay buffer. Then, for the next 5000 iterations, in addition to adding new transitions (according to the current policy) to the replay buffer, we also randomly sample a batch of transitions from the replay buffer and compute the modified detailed balance loss described in section 2. At the beginning of each trajectory, we also randomly pick an observation x from a batch of samples $\{(x_i, z_i)\}_{i=1}^{100}$ from the ground truth Markov network and we condition the GFlowNet on it. This is because we expect the model to learn to sample from $\mathbb{P}(z|x)$, so the sampled values for the latent variables should be different for different inputs x.

Note that, in the above, a **transition** is considered to be the high-level action of sampling all the values within a clique, while an **iteration** is the sampling of one node's value.

Finally, our code is based on a skeleton from the repository used in (Deleu et al., 2022).

3.3. Hyperparameter Selection

We tried different combinations of hyperparameters for the optimization (such as the learning rate and the optimizer) as well as for the model architecture (embedding dimensions, number of layers, self-attention heads and dimensions of the key vector) on a specific random graph. We then trained the GFlowNet with the best hyperparameters previously found on the two **new** graphs that are described in section 3.5 to carry out our empirical validation. Hence, we did not tune our hyperparameters on the same graphs that we used to test our hypotheses in section 3.5.

3.4. Evaluation Metrics

In addition to the training loss, we measure the discrepancy between the true posterior distribution $\mathbb{P}(z \mid x)$ and the forward policy P_F learned by the GFlowNet.

Kullback–Leibler (KL) divergence In our simplified setup where the true posterior can be computed tractably, we consider the conditional reverse KL divergence between the sampling distribution of the nodes values defined by the GFlowNet and the distribution under the true Markov network. More precisely, we compute

$$KL[P_F(. \mid x)||\mathbb{P}(. \mid x)] = \mathbb{E}_{z\sim P_F(z|x)} \log \frac{P_F(z \mid x)}{\mathbb{P}(z \mid x)}$$

where $P_F(z \mid x) = \sum_{\tau \in T(s_z)} P_F(\tau \mid x)$ is the distribution over z obtained by sampling trajectories from the GFlowNet, and $\mathbb{P}(z \mid x)$ is given in closed form by

$$\mathbb{P}(z \mid x) = \frac{\exp\left[-\mathcal{E}(x, z)\right]}{\sum_{z'} \exp\left[-\mathcal{E}(x, z')\right]} \tag{16}$$

Note that the above quantity is intractable to compute in general due to the summation over z' in the denominator. However, it is tractable in our case because we work with a small number of nodes with binary values. We perform a Monte-Carlo estimation of the outer expectation by drawing N trajectories τ_i each ending in z_i from the GFlowNet forward policy P_F, and then calculating

$$\widehat{KL} = \frac{1}{N} \sum_{i=1}^{N} [\log \frac{P_F(\tau_i \mid x)}{P_B(\tau_i \mid x)} - \log \mathbb{P}(z_i \mid x)]$$

We use $N = 16$ in our experiments.

Marginal data likelihood Another evaluation metric that is less sensitive to biases in the forward policy P_F is the marginal likelihood of the observed data x from a held out dataset sampled from the ground truth Markov network. When the GFlowNet loss, e.g., the modified detailed balance loss for modular energy functions, is minimized, we have

$$P_F(x) = F(s_0 \mid x) \sum_{\tau \ni x} P_F(\tau \mid x) = F(s_0 \mid x)$$

where s_0 is the initial GFlowNet state without any instantiated latent values. By definition, we have

$$\mathbb{P}(x) = \frac{\sum_z \exp[-\mathcal{E}(x, z)]}{Z}, \tag{17}$$

where Z is the partition function of the true Markov network. When we have minimized the GFlowNet loss, i.e., $P_F(x) = F(s_x)$, we might not necessarily have covered all the modes in the energy function $\mathcal{E}$. As a result, we have

$$\begin{aligned} Z\mathbb{P}(x) &= \sum_z \exp[-\mathcal{E}(x, z)] \\ &\geq \sum_{\top} \exp[-\mathcal{E}(\top, x)] \\ &= F(s_0 \mid x) \end{aligned} \tag{18}$$

where $\top$ denotes all z reachable by P_F. Overall,

$$\mathbb{P}(x) \geq \frac{F(s_0 \mid x)}{Z}. \tag{19}$$

Therefore, if $F(s_0|x)/Z$ approaches or matches $\mathbb{P}(x)$, we will have modeled the posterior well. Note that even when the GFlowNet has converged, the loss is not necessarily zero, e.g., due to limited capacity. In that case, $F(s_0 \mid x)$ will be a noisy estimate of $\sum_{\top} \exp[-\mathcal{E}(\top, x)]$.

3.5. Empirical Results

To evaluate our proposed training objective empirically, we design two experiments: one with a fixed hard-coded graph and one with a random latent graph. We show empirically that our model successfully learns to model the distribution over the values of the latent variables in the underlying graph in both cases. However, the second case reveals some interesting challenges that we leave to future work.

3.5.1. Fixed Hard-coded Graph

We start with training our model to predict the values of a fixed latent graph given the observation sampled from a ground truth Markov Network shown in Fig 2 with hard-coded energy functions. The energy function is designed in such a way that each clique c_i corresponds to a specific observation x. In particular, when all the observation nodes are equal to zero, clique c_1 is active with a high potential value corresponding to $(z_1, z_2, z_3) = (0, 0, 0)$. When all the observation nodes are equal to one, clique c_2 should be active with a high potential value corresponding to $(z_3, z_4) = (1, 1)$. Finally, if $(x_1, x_2, x_3) = (0, 0, 1)$, clique c_3 should be active with a high potential value corresponding to $(z_4, z_5, z_6) = (0, 0, 1)$. In all these three cases, the potential value is equal to 2 while any other value combinations of the latent and the observation variables have a small potential value equal to 0.01. As shown in Fig 3 (a), our model's loss and the conditional reverse KL divergence approach to almost zero while marginal data likelihood converges to the true likelihood from the ground truth Markov Network.

3.5.2. Random Graph

To test our method in a more general case, we define the ground truth Markov network as described in section 3.1

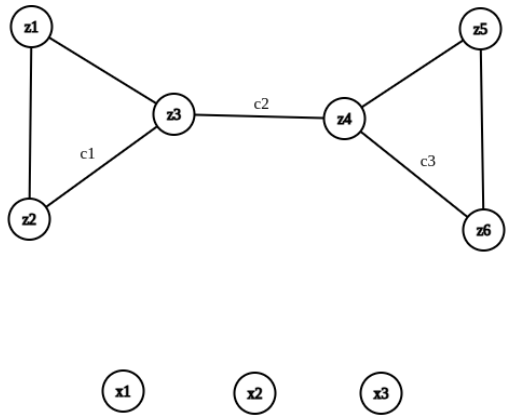
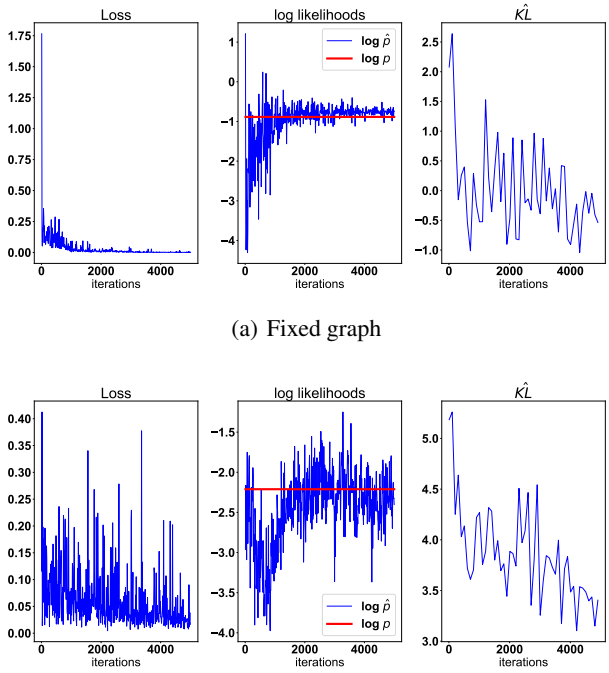


Figure 2. Fixed sparse graph. To avoid having an overcrowded graph visualization, the edges between observation nodes x_i themselves and with the latent nodes z_i are not shown.

with random energy functions. We set the dimensions of the observed and latent variables to be the same as in the fixed hard-coded graph case to allow for a fair comparison between the two experiments. The edges between latent variables are sampled from a Bernoulli distribution with parameter $p = 0.6$. Note that, we can control the level of the sparsity of the latent graph by changing p. As shown in Fig 3 (b), we observe that the curves are noisier compared with the previous setting with fixed hard-coded graph and the estimated reverse KL divergence stays far from 0 for the same number of iterations. However, $\log \hat{p}$ still seems to converge to the true log-likelihood.

3.6. Discussion

It is interesting to see that, despite the two graphs having the same dimensions, the same GFlowNet with the same hyperparameters has a noticeably worse training performance on the random graph. We can inspect the exact random graph that was generated for that specific experiment in figure 4. Not only there is a clique of size 4 which makes the dependencies more complicated, but the potential functions themselves are less structured (since they are random) than the ones defined in the hardcoded graph. It is thus not a surprise to see the performance degrade. However, one difficulty that we had is that even increasing the model's capacity (by increasing the embedding dimensions or the number of layers, for instance) didn't help in getting a better performance on random graphs. This is left for future investigations, but one hypothesis that can explain this is that when generating random potential functions, the model is essentially tasked to learn structured noise, and this can become quickly intractable as the cliques grow larger, which is one of the reasons why the inductive bias of sparsity is so important.



(a) Fixed graph

(b) Random graph

Figure 3. Latent graph value prediction results for the fixed graph (top) and random graph (bottom). Note that $\widehat{KL}$ can be negative since it's only a Monte-Carlo estimate. Moreover, $\log \hat{p}$ is the lower bound presented in Eq. 19 and $\log p$ is the true marginal log-likelihood $\log \mathbb{P}(x)$.

4. Conclusion and Future Work

We propose a new internal consistency training objective based on the detailed balance condition for GFlowNet training that exploits modularity in the target energy function. This new objective does not rely on complete trajectories for credit assignment, which makes it possible to apply GFlowNets to scenarios where complete trajectories are too expensive to obtain, e.g., in the training of the simplifed thinking machine. We tested our proposed training objective on synthetic data generated from a handcrafted Markov network and a randomly sampled one. The effectiveness of the new objective was evaluated by comparing the posterior distribution learned by the amortized sampler and the ground truth using the reverse KL divergence and the marginal data likelihood on a held-out dataset. Our results indicate that we are able to successfully learn an amortized sampler from $p(z|x)$ on small graphs with this new objective.

There are many future directions, both in terms of the learning algorithm and downstream applications. In this work, we have fixed the order of generation, e.g., fixing $P_B(s' \mid s) = 1$. This limits the GFlowNet's ability to choose different orders of completion for different input x. In addition, it is possible to train a separate "clique selection"

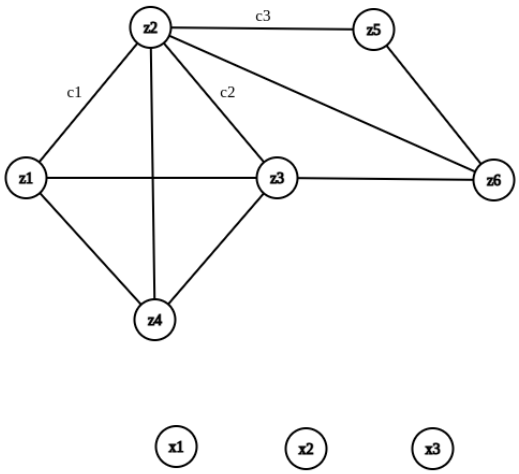


Figure 4. Random graph. To avoid having an overcrowded graph visualization, the edges between observation nodes x_i themselves and with the latent nodes z_i are not shown.

policy to prioritize cliques with high mutual information with the input x. In terms of building the simplified thinking machine, we note that the ground-truth Markov network is not provided. Instead, the parameters of such a network, i.e., the energy function, need to be learned jointly with the amortized sampler. Finally, we would like to eventually scale up our approach to realistic scenarios where there are potentially millions of latent variables that are sparsly connected (and where we might never visit a terminal state), which could find applications in areas such as gene regulatory networks.

References

Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

Bengio, E., Jain, M., Korablyov, M., Precup, D., and Bengio, Y. Flow network based generative models for non-iterative diverse candidate generation. *Neural Information Processing Systems (NeurIPS)*, 2021a.

Bengio, Y., Lahlou, S., Deleu, T., Hu, E., Tiwari, M., and Bengio, E. GFlowNet foundations. *arXiv preprint 2111.09266*, 2021b.

Deleu, T., Góis, A., Emezue, C., Rankawat, M., Lacoste-Julien, S., Bauer, S., and Bengio, Y. Bayesian structure learning with generative flow networks. *arXiv preprint arXiv:2202.13903*, 2022.

Goyal, A. and Bengio, Y. Inductive biases for deep learning of higher-level cognition. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 478(2266):20210068, 2022. doi: 10.1098/rspa.2021.0068. URL https://royalsocietypublishing.org/doi/abs/10.1098/rspa.2021.0068.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *International Conference on Machine Learning (ICML)*, 2018.

Hennigan, T., Cai, T., Norman, T., and Babuschkin, I. Haiku: Sonnet for JAX, 2020. URL http://github.com/deepmind/dm-haiku.

Malkin, N., Jain, M., Bengio, E., Sun, C., and Bengio, Y. Trajectory balance: Improved credit assignment in GFlowNets. *Neural Information Processing Systems (NeurIPS)*, 2022.

Sherrington, D. and Kirkpatrick, S. Solvable model of a spin-glass. *Phys. Rev. Lett.*, 35:1792–1796, Dec 1975. doi: 10.1103/PhysRevLett.35.1792. URL https://link.aps.org/doi/10.1103/PhysRevLett.35.1792.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.